PicAxe M2 - Das Speicherkompendium

Alles über RAM, Eeprom & Lookup-Table der PicAxe M2 Typen

Ausgabe 03-2017

Matthias Heuschele / SSE

Inhaltsverzeichnis

Der PicAxe M2 Speicher - Typen und Größen

Alle PixAce M2-Typen verfügen über zwei unterschiedliche Speichertypen; das RAM und das Flash-Eeprom, das im folgenden nur als Eeprom bezeichnet wird. Bestimmte PicAxe M2-Typen verfügen noch über einen dritten Speichertyp, der im Benutzerhandbuch als *„programm memory lookup - table“* bezeichnet wird, und den ich hier im weiteren mit *„lookup table“* oder einfach nur LUT bezeichnen werde.

In nachfolgender Tabelle finden Sie eine Übersicht, welche Speichergrößen und Speichertypen bei den einzelnen PicAxe M2-Typen zur Anwendung kommen:

PicAxe	***RAM***	***Eeprom***	***LUT***
08M2	*128 Bytes*	*256 Bytes*[1]	*N/A*
14M2	*512 Bytes*	*256 Bytes*	*512 Bytes*
18M2	*128 / 256 / 512 Bytes*[2]	*256 Bytes*[1]	*512 Bytes*
18M2+	*512 Bytes*	*256 Bytes*	*512 Bytes*
20M2	*512 Bytes*	*256 Bytes*	*512 Bytes*

Tabelle 1

Bevor ich auf die einzelnen Speichertypen detailliert eingehen werde, möchte ich Ihnen hier in einer kurzen Übersicht den Verwendungszweck der einzelnen Speichertypen aufzeigen:

Speichertyp	***Beschreibung***	***Zugriffsart***	***Verwendung***
RAM	*Speicher für flüchtige Daten*	*r/w*	*Programmvariablen*
Eeprom	*Speicher für nichtflüchtige Daten*	*r/w*	*Dynamische Daten*
LUT	*Speicher für nichtflüchtige Daten*	*ro*	*Statische Daten*

Tabelle 2

1 Bei den PicAxe Typen 08M2 und älteren 18M2 stehen unter Umständen nicht die kompletten 256 Bytes des Eeproms zur Verfügung. Bei diesen Typen ist die freie Eeprom Speicherkapazität von der der Größe des Programms abhängig.

2 Bei älteren 18M2 nur 256 Bytes RAM. Wenn bei diesem Typ mehr als einen Task verwendet wird, reduziert sich der verfügbare Speicher auf 128 Bytes.

RAM

Das RAM ist ein flüchtiger Speicher, dessen Inhalt nur solange besteht, bis die Spannungsversorgung der PicAxe unterbrochen wird. Im RAM werden Daten, die nur zur Laufzeit des Programms benötigt werden, gespeichert. Die ersten 28 Bytes (Byte 0-27) des RAMs werden von den allgemeinen Variablen belegt, d.h. die allgemeinen Variablen werden im RAM beginnend ab Adresse 0 bis Adresse 27 gespeichert. Das restliche RAM kann vom Programm frei verwendet werden. Auf dieses RAM kann z.B. mittels PEEK und POKE zugegriffen werden. Nachfolgende Tabelle zeigt den RAM-Aufbau der einzelnen M2-Typen.

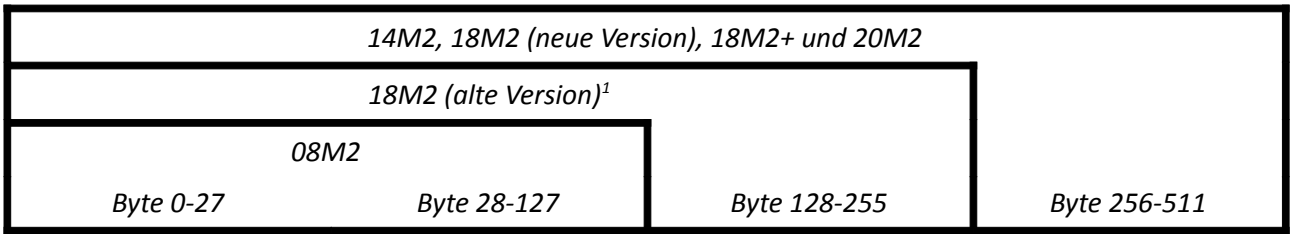

Tabelle 3

Allgemeine Variablen

8-Bit Variablen

Die PicAxe M2-Typen verfügen insgesamt über 28 Byte-Variablen, die jeweils 8 Bit (1 Byte) aufnehmen können. Jede Byte-Variable kann einen ganzzahligen positiven Wert zwischen 0 und 255 aufnehmen. In einem Basic-Programm werden diese 8 Bit-Variablen mit den Namen b0, b1, b2, (…), b26, b27 angesprochen, wobei das b im Variablenname für Byte steht.

16-Bit Variablen

Jeweils zwei aufeinanderfolgende Byte-Variablen können zu einer 16 Bit Variablen (Word-Variable) mit einem Low- und Highbyte zusammengefasst werden. Somit können die Kombinationen der Byte-Variablen b0/b1, b2/b3, (…), b24/b25, b26/b27 als jeweils eine Word-Variable verwendet werden. Jede Word-Variable kann einen ganzzahligen positiven Wert zwischen 0 und 65535 aufnehmen. In einem Basic-Programm werden diese 16 Bit-Variablen mit den Namen w0, w1, w2, (…), w12, w13 angesprochen, wobei das w im Variablenname für Word steht.
Byte- und Word-Variablen können NICHT unabhängig voneinander betrachtet werden, denn eine Änderung an einer Byte-Variablen ändert gleichzeitig auch den Wert der entsprechenden Word-Variablen. Entsprechend wird der Wert einer Word-Variablen geändert, sobald sich der Wert einer Byte-Variablen ändert, aus der sich die Word-Variable zusammensetzt.
Word-Variablen werden im RAM im Little-Endian Format gespeichert, d.h. zuerst das niederwertige Byte, dann das höherwertige Byte. Enthält z.B. die Variable w0 den Wert 0x11FF, wird im RAM zuerst 0xFF und dann 0x11 abgelegt. Somit enthielte Variable b0 dem Wert 0xFF und variable b1 dem Wert 0x11.

1 Sollte ein Programm mehr als einen Task verwendet, reduziert sich der verfügbare Speicher auf nur 128 Bytes.

Word-Variable	*High-Byte*	*Low-Byte*
w0	*b0*	*b1*
w1	*b2*	*b3*
w2	*b4*	*b5*
w3	*b6*	*b7*
w4	*b8*	*b9*
w5	*b10*	*b11*
w6	*b12*	*b13*
w7	*b14*	*b15*
w8	*b16*	*b17*
w9	*b18*	*b19*
w10	*b20*	*b21*
w11	*b22*	*b23*
w12	*b24*	*b25*
w13	*b26*	*b27*

Tabelle 4

1-Bit Variablen

Die ersten 32 Bits im RAM der M2-PicAxe können als 1 Bit-Variablen verwendet werden. In einem Basic-Programm werden diese 1 Bit-Variablen mit den Namen bit0, bit1, bit2, (…), bit31, bit32 angesprochen.

Mit den 1-Bit-Variablen verhält es sich entsprechend der Kombination von Byte und Word-Variablen. Die 1 Bit-Variablen 0 bis 7 entsprechen der Byte-Variablen b0. Die 1 Bit-Variablen 8 bis 15 entsprechen der Byte-Variablen b1, usw. Vom Standpunkt der Word-Variablen betrachtet, entsprechen die 1 Bit-Variablen 0 bis 15 der Word-Variable w0 und die 1 Bit-Variablen 16-31 der Word-Variablen w1. Auch hierbei ist wieder zu beachten, das jede Wertänderung einer 1 Bit-Variablen, eine Wertänderung der entsprechenden Byte- und Word-Variablen zufolge hat. Im Umgekehrten Fall kann jede Wertänderung einer Word oder Byte-Variablen entsprechend auch eine Wertänderung einer 1 Bit-Variablen zur Folge haben.

Word-Variable	*Byte-Variable*	*Bit-Variable*
w0	*b0, b1*	*bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7*
w1	*b2, b3*	*bit8, bit9, bit10, bit11, bit12, bit13, bit14, bit15*
w2	*b4, b5*	*bit16, bit17, bit18, bit19, bit20, bit21, bit22, bit23*
w3	*b6, b7*	*bit24, bit25, bit26, bit27, bit28, bit29, bit30, bit31*

Tabelle 5

Im folgenden eine Übersicht, wie und wo die einzelnen Variablen im RAM abgelegt werden, sowie noch einmal die Zusammensetzung der einzelnen Variablentypen:

RAM-Adresse	*Bit-Variable*	*Byte-Variable*	*Word-Variable*
0	*bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7*	*b0*	*w0*
1	*bit8, bit9, bit10, bit11, bit12, bit13, bit14, bit15*	*b1*	
2	*bit16, bit17, bit18, bit19, bit20, bit21, bit22, bit23*	*b2*	*w1*
3	*bit24, bit25, bit26, bit27, bit28, bit29, bit30, bit31*	*b3*	
4		*b4*	*w2*
5		*b5*	
6		*b6*	*w3*
7		*b7*	
8		*b8*	*w4*
9		*b9*	
10		*b10*	*w5*
11		*b11*	
12		*b12*	*w6*
13		*b13*	
14		*b14*	*w7*
15		*b15*	
16		*b16*	*w8*
17		*b17*	
18		*b18*	*w9*
19		*b19*	
20		*b20*	*w10*
21		*b21*	
22		*b22*	*w11*
23		*b23*	
24		*b24*	*w12*
25		*b25*	
26		*b26*	*w13*
27		*b27*	

Tabelle 6

Zeigervariable „bptr“

Alle M2 Typen verfügen über eine interne Zeigervariable mit dem Namen @bptr (Abkürzung für Bytepointer), mit der indirekt auf das RAM zugegriffen werden kann. Diese Zeigervariable wird nicht im Benutzer-RAM abgelegt, sondern ist fest in der PicAxe integriert. Der Namen der Zeigervariablen bptr ist etwas unglücklich gewählt, da man annehmen könnte, die Variable würde ein Byte enthalten, das auf eine Adresse im RAM zeigt. Wenn das so wäre, könnte man allerdings nur 256 Adressen ansprechen. Stattdessen enthält die Variable bptr eine WORD-Adresse, die auf eine beliebige Adresse im RAM zeigt. Stellt man der Variablen bptr noch ein @ voran, also @bptr, erhält man das Byte, das an der Adresse gespeichert ist, auf die die Variable bptr zeigt. Somit erklärt sich jetzt auch der Name Bytepointer.

Die Variable bptr zeigt auf eine Adresse, die ein Byte enthält.

Zum besseren Verständnis ein noch zwei Beispiele:

Beispiel 1:

Befehl	***Erklärung***
bptr = 300	*bptr die RAM-Adresse 300 zuweisen*
b0 = @bptr	*Den Inhalt von Adresse 300 nach Variable b0 kopieren*
@bptr = 20	*Wert 20 an der Adresse 300 speichern*

Beispiel 2:

Im RAM stehen z.B. folgende Byte-Werte ab Adresse 405:

Adresse	*405*	*406*	*407*	*408*
Inhalt (Byte)	*34*	*88*	*125*	*255*

Befehl	***Erklärung***
bptr = 404	*bptr die RAM-Adresse 404 zuweisen*
for b0 = 1 to 4	*Schleife mit 4 Wiederholungen*
b1 = @bptrinc	*Inhalt der Adresse auf die bptr zeigt, nach b0 kopieren und bptr um 1 erhöhen*
next b0	*Schleife wiederholen (4x)*

Durchgang	*0*	*1*	*2*	*3*	*4*
bptr	*404*	*405*	*406*	*407*	*408*
@bptr	*??*	*34*	*88*	*125*	*255*
b1	*??*	*34*	*88*	*125*	*255*

Da man mit der Variablen bptr auf jede beliebige RAM Adresse zugreifen kann und die allgemeinen Variablen auch im RAM ab Adresse 0 liegen, kann man mit bptr auch auf die allgemeinen Variablen zugreifen. Mit *bptr = 10 : @bptr = 99* weist man z.B. der Variablen b10 den Wert 99 zu, da Variable b10 im RAM an der Speicherstelle 10 steht (siehe Tabelle 6).

Systemvariable

Zusätzlich zu den allgemeinen Variablen verfügen die M2-Typen über weitere acht 16 Bit Variablen (Word-Variable). Dies Systemvariablen werden nicht im Benutzer-RAM abgelegt, sondern sind fest in der PicAxe integriert. Die 16 Bit Systemvariablen sind im Gegensatz zu den allgemeinen Variablen nicht in zwei 8-Bit Variablen (Low- Highbyte) bzw. sechzehn 1 Bit Variable unterteilt, sondern werden immer als komplette Word-Variable angesprochen. In einem Basic-Programm werden diese 16 Bit-Variablen mit den Namen s_w0, s_w1, s_w2, usw. angesprochen, wobei das s im Variablenname für System steht.

Die Systemvariablen sind ursprünglich für das PicAxe Betriebssystem reserviert und nicht für die Verwendung innerhalb eines Benutzerprogramms vorgesehen. Bisher werden aber nur die beiden Systemvariablen s_w0 und s_w7 vom PicAxe Betriebsystem verwendet, und das auch nur unter folgenden Gegebenheiten:

- In Anwendungen, die zwei oder mehrere Tasks gleichzeitig ausführen, enthält die Systemvariable s_w0 die Nummer des gerade ausgeführten Tasks. Wenn eine Anwendung aus nur einem Task besteht, wird die Systemvariable s_w0 vom PicAxe Betriebssystem nicht genutzt und kann von der Anwendung verwendet werden.
- Die Systemvariable s_w7 enthält die bisherige Laufzeit des aktuellen Programms in Sekunden. Mit der Anweisung **DISABLETIME**, kann man diesen Zeitnehmer abschalten. Anschließend kann man nun auch die Systemvariable s_w7 in der Anwendung verwenden. **DISABLETIME** funktioniert allerdings nur dann, wenn das Programm aus nur einem einzigen Task besteht. In Multitasking Programmen funktioniert **DISABLETIME** also nicht.

Anzumerken wäre noch, dass die beiden Systemvariablen s_w0 und s_w7 auch unter den Aliase *task* und *timer* anstatt s_w0 und s_w7 angesprochen werden können. So entspricht beispielsweise die Anweisung *s_w0 = 13* der Anweisung *task = 13* und *s_w7 = 11* der Anweisung *timer = 11.*

Folgende Tabelle zeigt noch einmal eine Übersicht aller Systemvariablen, deren Aliase , sowie unter welchen Gegebenheiten die beiden Systemvariablen s_w0 und s_w7 von der Anwendung verwendet werden können.

		Verwendbar im Programmen vom Typ	
Systemvariable	***Alias***	***Singletasking***	***Multitasking***
s_w0	*task*	*Ja*	*Nein*
s_w1		*Ja*	*Ja*
s_w2		*Ja*	*Ja*
s_w3		*Ja*	*Ja*
s_w4		*Ja*	*Ja*
s_w5		*Ja*	*Ja*
s_w6		*Ja*	*Ja*
s_w7	*timer*	*Ja, mit DISABLETIME*	*Nein*

Tabelle 7

Speicherzugriff mit Peek und Poke

Poke

Mittels Poke können Daten ins RAM geschrieben werden. Der Syntax des Poke Befehls lautet:

poke *Adresse*,
{ *Byte-Konstante* | *Byte-Variable* | *Word-Variable* | W*ord-Konstante* | **word** *Word-Variable* } [, { … }]

Adresse bezeichnet eine allgemeine Variable, Systemvariable oder Konstante, welche die Speicheradresse am RAM enthält, ab der die Daten geschrieben werden. Als Adresse kann Grundsätzlich jede Speicheradresse im RAM angegeben werden, sofern dieses nicht außerhalb des verfügbaren RAMs liegt. (siehe Tabelle 1).
Die Daten, die in das RAM geschrieben werden sollen, können als Byte-Konstante, Byte-Variable, Word-Variable, oder Word-Konstante angegeben werden. Die Anweisung **word** ist nur in Verbindung mit Word-Variablen erlaubt.
Bei Angabe einer Word-Variablen oder Word-Konstante ohne die Anweisung **word**, wird nur das niederwertigste Byte (Low-Byte) in das RAM geschrieben. Als Word-Variable kann entweder eine allgemeine Variable oder eine Systemvariable verwendet werden.
Eine mit Poke gespeicherte Word-Variablen wird im RAM im Little-Endian Format (Low-Byte, High-Byte) abgelegt.
Sollen mehrere Daten mit einer Poke Anweisung geschrieben werden, müssen diese durch ein Komma voneinander getrennt angegeben werden.

Im folgenden noch ein paar Beispiele:

Befehl	***Beschreibung***	***Anmerkung***
Poke 20, 1	*Wert 1 nach Adresse 20*	*Ok*

Befehl	*Beschreibung*	*Anmerkung*
Poke 55, b1	*Inhalt von b1 an Adresse 55 speichern*	*Ok*
Poke 200, b5, 4, b7, 99 ,0x100	*Inhalt von b5, Wert 4, Inhalt von b7, Wert 99, Wert 0x100 ab Adresse 55 speichern*	*Ok,*
Poke b4, b5	*Inhalt von b5 an der Adresse die in b4 steht, speichern*	*Ok*
Poke w1, 0x67, b10	*Wert 0x67 und Inhalt von b10 an der Adresse die in w4 steht speichern*	*Ok*
Poke 243, word w0	*Inhalt von w0 ab der Adresse die in w4 steht speichern*	*Ok*
Poke s_w3, w1	*Low-Byte von w1 an der Adresse die in s_w3 steht, speichern*	*Nur das Low-Byte wird gespeichert, da die Anweisung **word** fehlt*
Poke 30, word 0x1234	*Ungültig*	*Word-Konstante in Verbindung mit der Anweisung **word** ist nicht zulässig*
Poke 30, 0x1234	*Low-Byte (0x34) an Adresse 30 speichern*	*Nur das Low-Byte wird gespeichert*
Poke 100, word b1	*Ungültig*	*B1 ist keine Word-Variable*
Poke Base, word Speed	*Inhalt von Variable Speed an der Adresse die in Base steht speichern*	*Ok, sofern Speed eine Word-Variable ist*

Tabelle 8

Da man mit Poke auf jede beliebige RAM Adresse zugreifen kann und die allgemeinen Variablen auch im RAM ab Adresse 0 liegen, kann man mit Poke auch auf die allgemeinen Variablen zugreifen. Mit *Poke 4, 22* weist man z.B. der Variablen b4 den Wert 22 zu, da Variable b4 im RAM an der Speicherstelle 4 steht (siehe Tabelle 6). Gleichzeitig weist man mit dieser Anweisung dem High-Byte von w0 den Wert 22 zu (siehe Tabelle 4).

Peek

Mittels Peek können Daten aus dem RAM gelesen und in einer Variablen gespeichert werden. Der Syntax des Peek Befehls lautet:

peek Adresse, { *Byte-Variable* | *Word-Variable* | **word** *Word-Variable* } [, { … }]

Adresse bezeichnet eine allgemeine Variable, Systemvariable oder Konstante, welche die Speicheradresse im RAM enthält, ab der die Daten gelesen werden. Als Adresse kann Grundsätzlich jede Speicheradresse im RAM angegeben werden, sofern dieses nicht außerhalb des verfügbaren RAMs liegt. (siehe Tabelle 1).

Die zu lesenden Daten können in einer Byte-Variablen oder Word-Variablen gespeichert werden. Möchte man ein Word aus dem RAM in einer Word-Variablen speichern, muss vor der Word-Variablen die Anweisung **word** angegeben werden. Wird dieses nicht angegeben, wird nur das Byte, das an der angegebenen Adresse steht, in der Word-Variablen gespeichert. Als Word-Variable kann entweder eine allgemeine Variable oder eine Systemvariable verwendet werden. Ein Wert, der in einer Word-Variablen gespeichert werden soll, wird im Little-Endian Format aus dem RAM gelesen (Low-Byte = Adresse, High-Byte = Adresse +1)

Sollen mehrere Variablen mit einer Peek Anweisung gelesen werden, müssen diese durch ein Komma voneinander getrennt angegeben werden.

Im folgenden noch ein paar Beispiele:

Befehl	***Beschreibung***	***Anmerkung***
Peek 20, b1	*Inhalt von Adresse 20 nach b1 kopieren*	*Ok*
Peek 55, b1, b2, b3	*Inhalt von Adresse 55, 56 und 57 nach b1, b2 und b3 kopieren*	*Ok*
Peek b0, b1	*Inhalt von Adresse die in b0 steht, nach b1 kopieren*	*Ok*
Peek 100, word s_w1	*Inhalt von Adresse 100 nach Systemvariable s_w1 kopieren*	*Ok*
Peek 50, w1	*Inhalt (Byte) von Adresse 100 nach w1 kopieren*	*Nur ein einzelnes Byte wird gelesen, da die Anweisung **word** fehlt*
Peek w1, word b10	*Ungültig*	*b10 ist keine Word-Variable*
Peek base, word Speed	*Inhalt von Adresse die in base steht, nach Variable Speed kopieren.*	*Ok, sofernSpeed keine Byte-Variable oder Konstante*

Tabelle 9

Da man mit Peek auf jede beliebige RAM Adresse zugreifen kann und die allgemeinen Variablen auch im RAM ab Adresse 0 liegen, kann man mit Peeke auch auf die allgemeinen Variablen zugreifen. Mit *Peek b0, b1* weist man z.B. der Variablen b1 den Inhalt der Variablen b0 zu (b1 = b0) (siehe Tabelle 6). Gleichzeitig kopiert man mit dieser Anweisung das High-Byte von Variable w0 in das Low-Byte von w0 (siehe Tabelle 4).

Zugriffszeiten

Alle Zeitmessungen wurden auf einer PicAxe 14M2 - Firmware-Version 6.A und einer Standartfrequenz von 4 MHz durchgeführt. Die Zugriffszeiten sollen nicht als Referenz dienen, sondern zeigen nur die zeitliche Differenz der einzelnen Befehle zueinander.

Platz	*Befehl*	*Dauer in ms*
1	*Poke Bytevariable, Bytevariable*	*0,6*
2	*Poke Bytekonstante, Bytevariable*	*0,6*
3	*Poke Wordvariable, Bytevariable*	*0,7*
4	*Poke Wordkonstante, Bytevariable*	*0,8*
5	*Poke Bytekonstante, Wordvariable*	*1,3*
6	*Poke Wordkonstante, Wordvariable*	*1,3*
7	*Poke Bytevariable, Wordvariable*	*2,5*
8	*Poke Wordvariable, Wordvariable*	*2,5*

Tabelle 10

Typ	*Befehl*	*Dauer in ms*
1	*Peek Bytekvariable, Bytevariable*	*0,6*
2	*Peek Bytekonstante, Bytevariable*	*0,7*
3	*Peek Wordvariable, Bytevariable*	*0,7*
4	*Peek Wordkonstante, Bytevariable*	*0,7*
5	*Peek Bytekonstante, Wordvariable*	*1,3*
6	*Peek Wordkonstante, Wordvariable*	*1,5*
7	*Peek Bytevariable, Wordvariable*	*2,5*
8	*Peek Wordvariable, Wordvariable*	*2,6*

Tabelle 11

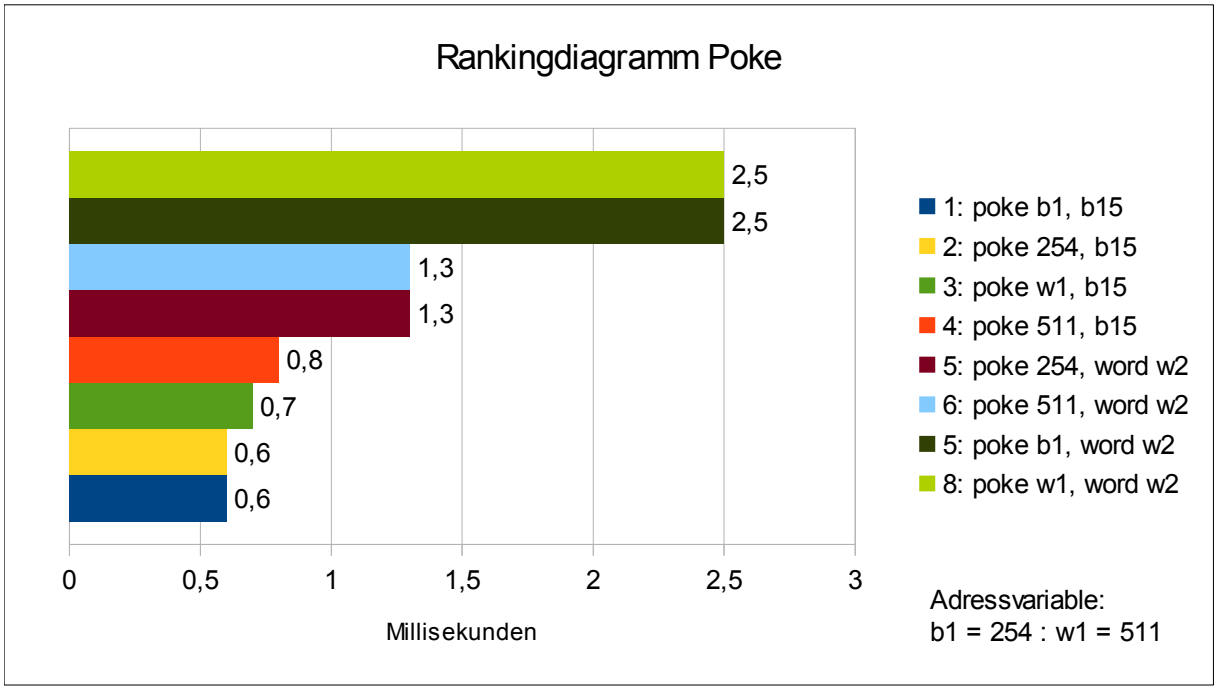

Abbildung 1

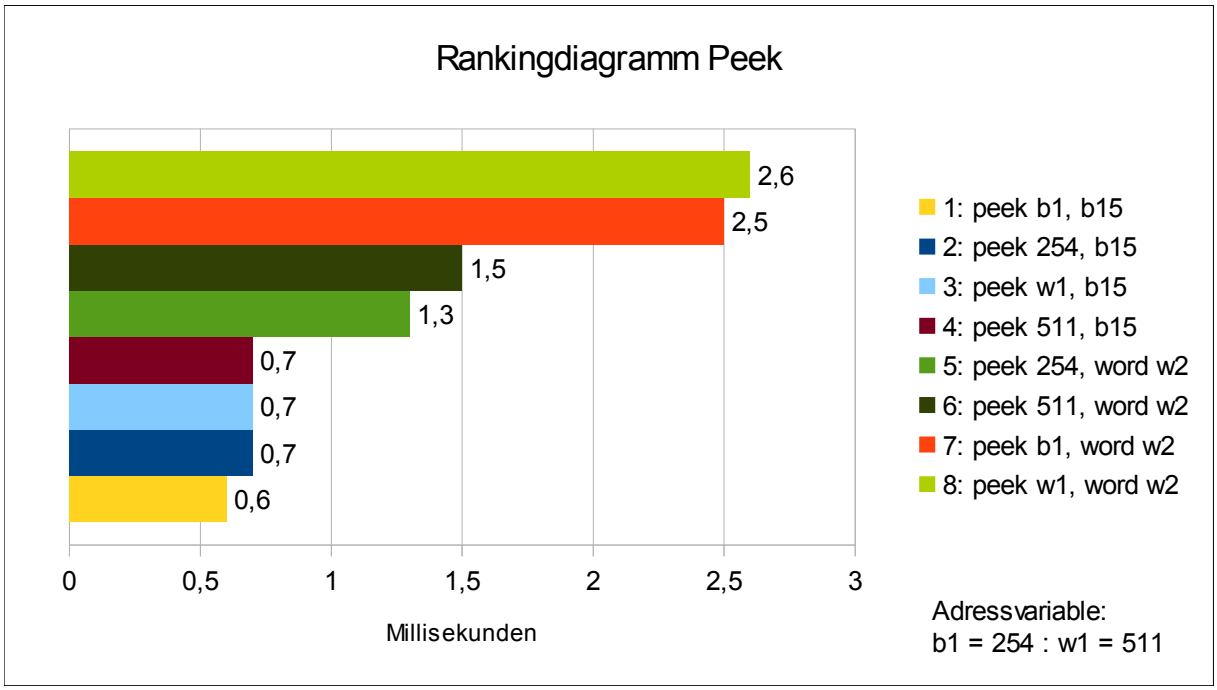

Abbildung 2

Eeprom

Im Eeprom der M2-PicAxe können Daten gespeichert werden, die im Gegensatz zum RAM auch noch nach dem ausschalten der Spannungsversorgung erhalten bleiben. Diese Daten können entweder zur Laufzeit des Programms in das Eeprom gespeichert werden, oder bereits während des Programmdownloads.

Wie bereits im ersten Kapitel besprochen, verfügen alle PicAxe M2-Typen über ein Eeprom mit einer Größe von 256 Bytes. Bei den PicAxe Typen 14M2, neuere 18M2 18M2+ und 20M2 ist das Eeprom unabhängig vom Programmspeicher.
Bei den PicAxe Typen 08M2 und älteren 18M2 liegt der 256 Byte umfassende Eeprom Speicher innerhalb, bzw. am Ende des 2048 Bytes umfassenden Programmspeicher, d.h. Programm-speicher und Eeprom teilen sich bei diesen beiden PicAxe-Typen den gesamten Bereich von 2048 Bytes. Wenn nun ein Programm mehr als 1792 Bytes (2048 Bytes - 256 Bytes Eeprom) umfasst, verkleinert sich das zur Verfügung stehende Eeprom entsprechend. Wie groß das aktuelle Programm ist, kann man einfach ermitteln, in dem man innerhalb der Entwicklungsumgebung einen Syntax-Check durchführt, welcher auch die Programmgröße in Bytes anzeigt. Wenn die Programmgröße mehr als 1792 beträgt, ermittelt man die zur Verfügung stehende Eeprom Kapazität x durch die Formel x = 2048 - Programmgröße.

Um auf das Eeprom zuzugreifen stehen dem Programmierer drei Möglichkeiten zur Verfügung:

Anweisung / Befehl	***Verwendung***
Eeprom	*Statische Daten während des Programmdownloads im Eeprom speichern*
Write	*Dynamische Daten während des Programmablaufs im Eeprom speichern*
Read	*Daten während des Programmablaufs aus dem Eeprom lesen*

Tabelle 12

Speicherzugriff mit Eeprom, Write und Read

Eeprom

Möchte man während das Programmdownloads Daten, wie z.B. statischen Text, in das Eeprom schreiben, muss im Quellcode die Anweisung **Eeprom** oder die gleichberechtigte **Data** Anweisung, gefolgt von den zu speichernden Daten enthalten sein. Die Anweisung **Eeprom** bzw. **Data** hat folgenden Syntax:

eeprom | data [*Adresse*], (*Byte-Konstante* , { … })

Die optionale Adresse bezeichnet eine Byte-Konstante, welche die Startadresse im Eeprom enthält, ab der alle nachfolgenden Daten gespeichert werden. Als Adresse kann Grundsätzlich jede Speicheradresse im Eeprom angegeben werden, sofern sich diese nicht außerhalb des verfügbaren Eeprom Speicher befindet (siehe Tabelle 1).
Wenn in der ersten im Quelltext vorhanden Eeprom Anweisung, keine Adresse angegeben, wird automatisch ab Adresse Null gespeichert. Mit jedem Byte, das im Eeprom gespeichert wird, erhöht sich automatisch die Speicheradresse um die Anzahl der zu speichernden Bytes. Folgen weitere Eeprom

Anweisungen ohne Adressangabe, werden die nachfolgenden Daten ab der aktuellen Adresse, gespeichert und die Adresse entsprechen der Anzahl zu speichernden Bytes wieder erhöht. Speicherbereiche, die einmal mit der Eeprom Anweisung beschrieben wurden, können nicht durch eine andere Eeprom Anweisung überschrieben werden. Folgende Tabelle soll das genannte noch einmal Verdeutlichen:

	Eeprom Adresse / Inhalt									
Anweisung (Folgend)	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***Anmerkung***
Eeprom (20, 22, 50)	*20*	*22*	*50*							*Adresszeiger ist jetzt 3*
Eeprom (11, 12, 13)	*20*	*22*	*50*	*11*	*12*	*13*				*Adresszeiger ist jetzt 6*
Eeprom 7, (98, 99)	*20*	*22*	*50*	*11*	*12*	*13*		*98*	*99*	*Adresszeiger ist jetzt 9*
Eeprom 6 (30, 31)	*20*	*22*	*50*	*11*	*12*	*13*		*98*	*99*	*Ungültig, Zelle 7 wurde schon beschrieben*
Eeprom 6, (30)	*20*	*22*	*50*	*11*	*12*	*13*	*30*	*98*	*99*	*Adresszeiger ist jetzt 7*
Eeprom (75)	*20*	*22*	*50*	*11*	*12*	*13*	*30*	*98*	*99*	*Ungültig. Zelle 7 wurde schon beschrieben*

Tabelle 13

In folgender Tabelle noch ein paar Beispiele für gültige und ungültige Adressangaben:

Anweisung	***Anmerkung***
Eeprom 1, (20)	*Ok: 1 ist eine Byte-Konstante*
Eeprom b1, (20)	*Ungültig: b1 ist eine Variable und keine Byte-Konstante*
Symbol Base = 12 *Eeprom base, (20)*	*Ok, Base ist eine Byte-Konstante*
Symbol Base2 = 300 *Eeprom base2 (20)*	*Ungültig: Base2 ist **keine** Byte-Konstante, da Wert > 255*
Eeprom 600, (20)	*Ungültig: Adresse 600 liegt außerhalb des verfügbaren Eeprom Speicher*

Tabelle 14

Die Daten, die in das Eeprom geschrieben werden sollen, müssen als Byte-Konstante angegeben werden. Variablen sind nicht erlaubt. Prinzipiell könnten auch Word-Konstante angegeben werden, jedoch wird dabei nur das Low-Byte der Word-Konstante in das Eeprom geschrieben. Sollen mehrere Bytes in das Eeprom geschrieben werden, müssen diese durch ein Komma voneinander getrennt angegeben werden.

Als Byte-Konstante können auch die Ausdrücke wie **eeprom 0, („T“, „e“, „x“, „t“)** oder **eeprom 0, („Text“)** verwendet werden, wobei die zweite Anweisung eine Kurzform der ersten Anweisung darstellt. In beiden Fällen werden die einzelnen Zeichen als ASCII-Zeichen interpretiert und deren ASCII-Werte byteweise im Eeprom gespeichert. Der Ausdruck („T“, „e“, „x“, „t“) bzw. („Text“) entspräche somit dem Ausdruck (54, 65, 78, 74)

Im folgenden noch ein paar Beispiele:

Anweisung	*Anmerkung*
Eeprom 15, (99)	*Ok: Wert 99 an Adresse 15 speichern*
Eeprom 20, (b1)	*Ungültig: B1 ist eine Variable und keine Byte-Konstante*
Eeprom 10, („Nummer:“ , 13, 10)	*Ok: 7 ASCII Codes, Wert 13, Wert 10 ab Adresse 10 speichern*
Eeprom (0x99)	*Ok: Wert 0xFF an aktuelle Position speichern*
Eeprom 20, (0x11FF)	*Ok: Wert 0xFF an Adresse 20 speichern. 0x11 geht verloren*
Symbol Speed = b1 *Eeprom (Speed)*	*Ungültig: Speed ist eine Variable in keine Byte-Konstante*
Symbol Char = „A“ *Eeprom (Char)*	*Ok: Char ist eine Bytevariable („A“ = ASCII-Code 65)*
Eeprom 510, („Text“)	*Ungültig: Hier wird versucht auf Adresse 510 bis 513 zu schreiben. Maximale Adresse ist jedoch 512*

Tabelle 15

In manchen Fällen möchte man die in der Eeprom Anweisung enthaltenen Daten nicht mit dem Programmdownload übertragen, sondern die im Eeprom bereits enthaltenen Daten trotz Programmdownload beibehalten: Zum Beispiel könnte man im ersten Programmdownload mit der Eeprom Anweisung eine Seriennummer im Eeprom speichern. Wenn nun ein Programmupdate vorliegt, soll diese Nummer nicht überschrieben werden, sondern nur das Programm aktualisiert werden. Für diese Fälle steht die Pre-Processor Anweisung **#NO_DATA** zur Verfügung. Wenn sich diese Anweisung im Quelltext befindet, werden die Daten in den Eeprom Anweisungen beim Programmdownload **nicht** im Eeprom gespeichert und evtl. vorhandene Daten bleiben erhalten

Write

Daten wie z.B. Messwerte und Zustände, können während der Programmausführung mit dem **Write**-Befehl in das Eeprom geschrieben werden und somit nach einer Spannungsunterbrechung wieder aus dem Eeprom gelesen werden. Der Syntax des **Write**-Befehls entspricht dem **Poke**-Befehl und lautet:

write *Adresse*,

{ *Byte-Konstante* | *Byte-Variable* | *Word-Variable* | W*ord-Konstante* | **word** *Word-Variable* } [, { … }]

Adresse bezeichnet eine allgemeine Variable, Systemvariable oder Konstante, welche die Speicheradresse im Eeprom enthält, ab der die Daten geschrieben werden.
Als Adresse kann Grundsätzlich jeder beliebige Wert angegeben werden. Liegt die Speicheradresse ganz oder teilweise außerhalb des verfügbaren Eeprom Speichers, werden nur die Bytes gespeichert, die sich innerhalb des verfügbaren Eeprom Speicher befinden. Bei einem Programmtest innerhalb der Entwicklungsumgebung wird dabei keine Fehlermeldung ausgegeben.

Die Daten, die in das RAM geschrieben werden sollen, können als Byte-Konstante, Byte-Variable, Word-Variable, oder Word-Konstante angegeben werden. Die Anweisung **word** ist nur in Verbindung mit Word-

Variablen erlaubt.
Bei Angabe einer Word-Variablen oder Word-Konstante ohne die Anweisung **word**, wird nur das niederwertigste Byte (Low-Byte) in das Eeprom geschrieben. Als Word-Variable kann entweder eine allgemeine Variable oder eine Systemvariable verwendet werden.
Eine mit Write gespeicherte Word-Variablen wird im Eeprom im Little-Endian Format (Low-Byte, High-Byte) abgelegt.
Sollen mehrere Daten mit dem Write Befehl geschrieben werden, müssen diese durch ein Komma voneinander getrennt angegeben werden.

Im folgenden noch ein paar Beispiele:

Befehl	***Beschreibung***	***Anmerkung***
Write 20, 1	*Wert 1 nach Adresse 20*	*Ok*
Write 55, b1	*Inhalt von b1 an Adresse 55 speichern*	*Ok*
Write 200, b5, 4, b7, 99 ,0x100	*Inhalt von b5, Wert 4, Inhalt von b7, Wert 99, Wert 0x100 ab Adresse 55 speichern*	*Ok,*
Write b4, b5	*Inhalt von b5 an der Adresse die in b4 steht, speichern*	*Ok*
Write w1, 0x67, b10	*Wert 0x67 und Inhalt von b10 an der Adresse die in w4 steht speichern*	*Ok*
Write 243, word w0	*Inhalt von w0 ab der Adresse die in w4 steht speichern*	*Ok*
Write s_w3, w1	*Low-Byte von w1 an der Adresse die in s_w3 steht, speichern*	*Nur das Low-Byte wird gespeichert, da die Anweisung **word** fehlt*
Write 30, word 0x1234	*Ungültig*	*Word-Konstante in Verbindung mit der Anweisung **word** ist nicht zulässig*
Write 30, 0x1234	*Low-Byte (0x34) an Adresse 30 speichern*	*Nur das Low-Byte wird gespeichert*
Write 100, word b1	*Ungültig*	*B1 ist keine Word-Variable*
Write Base, word Speed	*Inhalt von Variable Speed an der Adresse die in Base steht speichern*	*Ok, sofern Speed eine Word-Variable ist*
Write 10000, b1	*Inhalt von b1 an Adresse 10000 speichern*	*Befehl wird ignoriert, da Adresse 10000 außerhalb des verfügbaren Eeprom Speicher liegt*

Befehl	*Beschreibung*	*Anmerkung*
Write 255, word w1	*Inhalt von w0 ab Adresse 255 speichern*	*Nur das Low-Byte von w1 wird an Adresse 255 gespeichert. Die Adresse des High-Byte liegt an Adresse 256 und somit außerhalb des verfügbaren Eeprom Speicher*

Tabelle 16

Read

Mittels Read können Daten aus dem Eeprom gelesen und in einer Variablen gespeichert werden. Der Syntax des Read-Befehls entspricht dem Peek-Befehl und lautet:

Read *Adresse*, { *Byte-Variable* | *Word-Variable* | **word** *Word-Variable* } [, { … }]

Adresse bezeichnet eine allgemeine Variable, Systemvariable oder Konstante, welche die Speicheradresse im Eeprom enthält, ab der die Daten gelesen werden.
Als Adresse kann Grundsätzlich jeder beliebige Wert angegeben werden. Liegt die Speicheradresse ganz oder teilweise außerhalb des verfügbaren Eeprom Speichers, werden nur die Bytes gelesen, die sich innerhalb des verfügbaren Eeprom Speicher befinden. Bei einem Programmtest innerhalb der Entwicklungsumgebung wird dabei keine Fehlermeldung ausgegeben.

Die zu lesenden Daten können in einer Byte-Variablen oder Word-Variablen gespeichert werden. Möchte man ein Word aus dem Eeprom in einer Word-Variablen speichern, muss vor der Word-Variablen die Anweisung **word** angegeben werden. Wird dieses nicht angegeben, wird nur das Byte, das an der angegebenen Adresse steht, in der Word-Variablen gespeichert. Als Word-Variable kann entweder eine allgemeine Variable oder eine Systemvariable verwendet werden. Ein Wert, der in einer Word-Variablen gespeichert werden soll, wird im Little-Endian Format aus dem RAM gelesen (Low-Byte = Adresse, High-Byte = Adresse +1)

Sollen mehrere Variablen mit dem Read Befehl gelesen werden, müssen diese durch ein Komma voneinander getrennt angegeben werden.

Im folgenden noch ein paar Beispiele:

Befehl	*Beschreibung*	*Anmerkung*
Read 20, b1	*Inhalt von Adresse 20 nach b1 kopieren*	*Ok*
Read 55, b1, b2, b3	*Inhalt von Adresse 55, 56 und 57 nach b1, b2 und b3 kopieren*	*Ok*
Read b0, b1	*Inhalt von Adresse die in b0 steht, nach b1 kopieren*	*Ok*
Read 20, word w0	*Inhalt von Adresse 20 nach Variable w0 kopieren*	*Ok*

Befehl	***Beschreibung***	***Anmerkung***
Read 100, word s_w1	*Inhalt von Adresse 100 nach Systemvariable s_w1 kopieren*	*Ok*
Read 50, w1	*Inhalt (Byte) von Adresse 100 nach w1 kopieren*	*Nur ein einzelnes Byte wird gelesen, da die Anweisung **word** fehlt*
Read w1, word b10	*Ungültig*	*b10 ist keine Word-Variable*
Read base, word Speed	*Inhalt von Adresse die in base steht, nach Variable Speed kopieren.*	*Ok, sofernSpeed keine Byte-Variable oder Konstante*
Read 10000, b1	*Inhalt von Adresse 10000 nach b1 kopieren*	*Befehl wird ignoriert, da Adresse 10000 außerhalb des verfügbaren Eeprom Speicher liegt*
Read 255, word w1	*Inhalt von Adresse 255 nach Variable w1 kopieren*	*Nur das Low-Byte von w1 wird von Adresse 255 gelesen. Die Adresse des High-Byte liegt an Adresse 256 und somit außerhalb des verfügbaren Eeprom Speicher*

Tabelle 17

Lookup-Table

Mit Ausnahme der Picaxe 08M2 verfügen alle anderen M2-Typen über einen 512 Bytes umfassenden Speicherbereich, der Lookup-Table oder kurz LUT genannt wird. In der LUT können Daten gespeichert werden, die wie im Eeprom auch nach dem ausschalten der Spannungsversorgung erhalten bleiben. Diese Daten können im Gegensatz zum Eeprom jedoch nicht zur Laufzeit des Programms gespeichert werden, sondern nur während des Programmdownloads. Die LTU wurde für die Speicherung von rein statischen Daten wie Texte, Konstante, Seriennummern, usw. konzipiert.

Um auf dies LUT zuzugreifen stehen dem Programmierer drei Möglichkeiten zur Verfügung:

Anweisung / Befehl	***Verwendung***
Table	*Statische Daten während des Programmdownloads in der LUT speichern*
Tablecopy	*Daten während des Programmablaufs aus der LUT in das RAM kopieren*
Readtable	*Daten während des Programmablaufs aus der LUT in Variablen kopieren*

Tabelle 18

Speicherzugriff mit Table, Tablecopy und Readtable

Table

Möchte man während das Programmdownloads Daten, wie z.B. statischen Text, in die LUT schreiben, muss im Quellcode die Anweisung **Table**, gefolgt von den zu speichernden Daten enthalten sein. Der Syntax der Anweisung **Table** entspricht der **Eeprom** Anweisung und hat folgenden Syntax:

table [*Adresse*], (*Byte-Konstante* | *Word-Konstante* , { … })

Die optionale Adresse bezeichnet eine Byte oder Word-Konstante, welche die Startadresse in der LUT enthält, ab der alle nachfolgenden Daten gespeichert werden. Als Adresse kann Grundsätzlich jede Speicheradresse in der LUT angegeben werden, sofern sich diese nicht außerhalb des verfügbaren LUT Speichers befindet. (siehe Tabelle 1).
Wenn in der ersten im Quelltext vorhanden LUT Anweisung, keine Adresse angegeben, wird automatisch ab Adresse Null gespeichert. Mit jedem Byte, das in der LUT gespeichert wird, erhöht sich automatisch die Speicheradresse um die Anzahl der zu speichernden Bytes. Folgen weitere Table Anweisungen ohne Adressangabe, werden die nachfolgenden Daten ab der aktuellen Adresse, gespeichert und die Adresse entsprechen der Anzahl zu speichernden Bytes wieder erhöht. Speicherbereiche, die einmal mit der LUT Anweisung beschrieben wurden, können nicht durch eine andere LUT Anweisung überschrieben werden. Folgende Tabelle soll das genannte noch einmal Verdeutlichen:

	LUT Adresse / Inhalt									
Anweisung (Folgend)	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***Anmerkung***
Table (20, 22, 50)	*20*	*22*	*50*							*Adresszeiger ist jetzt 3*
Table (11, 12, 13)	*20*	*22*	*50*	*11*	*12*	*13*				*Adresszeiger ist jetzt 6*
Table 7, (98, 99)	*20*	*22*	*50*	*11*	*12*	*13*		*98*	*99*	*Adresszeiger ist jetzt 9*
Table 6 (30, 31)	*20*	*22*	*50*	*11*	*12*	*13*		*98*	*99*	*Ungültig, Zelle 7 wurde schon beschrieben*
Table 6, (30)	*20*	*22*	*50*	*11*	*12*	*13*	*30*	*98*	*99*	*Adresszeiger ist jetzt 7*
Table (75)	*20*	*22*	*50*	*11*	*12*	*13*	*30*	*98*	*99*	*Ungültig. Zelle 7 wurde schon beschrieben*

Tabelle 19

Die Daten, die in die LUT geschrieben werden sollen, müssen als Byte-Konstante angegeben werden. Variablen sind nicht erlaubt. Prinzipiell könnten auch Word-Konstante angegeben werden, jedoch wird dabei nur das Low-Byte der Word-Konstante in die LUT geschrieben. Sollen mehrere Bytes in die LUT geschrieben werden, müssen diese durch ein Komma voneinander getrennt angegeben werden.

Als Byte-Konstante können auch die Ausdrücke wie **table 0, („T“, „e“, „x“, „t“)** oder **table 0, („Text“)** verwendet werden, wobei die zweite Anweisung eine Kurzform der ersten Anweisung darstellt. In beiden Fällen werden die einzelnen Zeichen als ASCII-Zeichen interpretiert und deren ASCII-Werte byteweise in der LUT gespeichert. Der Ausdruck („T“, „e“, „x“, „t“) bzw. („Text“) entspräche somit dem Ausdruck (54, 65, 78, 74)

Im folgenden noch ein paar Beispiele:

Anweisung	***Anmerkung***
Table 15, (99)	*Ok: Wert 99 an Adresse 15 speichern*
Table 500, (b1)	*Ungültig: B1 ist eine Variable und keine Byte-Konstante*
Table 300, („Nummer:“ , 13, 10)	*Ok: 7 ASCII Codes, Wert 13, Wert 10 ab Adresse 10 speichern*
Table (0x99)	*Ok: Wert 0xFF an aktuelle Position speichern*
Table 20, (0x11FF)	*Ok: Wert 0xFF an Adresse 20 speichern. 0x11 geht verloren*
Symbol Speed = b1 *Table (Speed)*	*Ungültig: Speed ist eine Variable in keine Byte-Konstante*
Symbol Char = „A“ *Table (Char)*	*Ok: Char ist eine Bytevariable („A“ = ASCII-Code 65)*
Table 600, (20)	*Ungültig: Adresse 600 liegt außerhalb des verfügbaren LUT Speicher*

Tabelle 20

Tablecopy

Mit dem **Tablecopy**-Befehl können Datenblöcke aus der LUT gelesen und in das RAM kopiert werden. Der Syntax des **Tablecopy**-Befehls lautet:

tablecopy A*dresse*, *Blocklänge*

Adresse bezeichnet eine allgemeine Variable, Systemvariable oder Konstante die angibt, ab welcher Adresse (Startadresse) die Daten aus der LTU gelesen und ab welcher Adresse (Zieladresse) die Daten im RAM gespeichert werden. Die Adresse ist somit gleichzeitig eine Start und Zieladresse.
Als Adresse kann Grundsätzlich jeder beliebige Wert angegeben werden. Liegt die Speicheradresse ganz oder teilweise außerhalb des verfügbaren LUT Speichers, werden nur die Bytes kopiert, die sich innerhalb des verfügbaren LUT Speicher befinden. Die restlichen Bytes außerhalb des LUT Speicher werden ignoriert. Bei einem Programmtest innerhalb der Entwicklungsumgebung wird dabei keine Fehlermeldung ausgegeben.

Die Blocklänge bezeichnet eine allgemeine Variable, Systemvariable oder Konstante, die die Anzahl der zu kopierenden Bytes beinhaltet. Befinden sich die zu kopierenden Daten ganz oder teilweise außerhalb des verfügbaren LUT Speicher, werden nur die Bytes in das RAM kopiert, die sich innerhalb des verfügbaren LUT Speicher befinden. Die restlichen Bytes außerhalb des LUT Speicher werden ignoriert. Auch hier wird innerhalb der Entwicklungsumgebung keine Fehlermeldung ausgegeben.

Bis auf die älteren 18M2, entspricht bei allen PicAxe M2-Typen[2] die Speichergröße der LUT der Speichergröße des RAMs, so dass hier keine Seiteneffekte auftreten können. Bei einer älteren 18M2 würde bei dem Befehl **tablecopy 250, 10** ein Seiteneffekt auftreten. Mit diesem Befehl sollen die Bytes ab Adresse 250 bis Adresse 259 (10 Bytes) aus der LUT gelesen und im RAM ab Adresse 250 gespeichert werden. Da eine ältere 18M2 aber nur 256 Bytes RAM beinhaltet, würde hier ein Fehler auftreten, da sich die Zieladresse außerhalb des verfügbaren RAMs befindet. Bei einer älteren 18M2 können daher nur die unteren 256 Bytes der LUT verwendet werden.

Mit dem **Tablecopy**-Befehl kann man die allgemeine Variablen teilweise oder komplett überschreiben, da diese wie bereits bekannt ab Adresse 0 im RAM liegen. Der Befehl tablecopy 1, 0 würde z.B. die Variable b1 überschreiben. Mit Tablecopy 0, 27 überschreibt man alle allgemeine Variablen (siehe Tabelle 6).

Im folgenden noch ein paar Beispiele:

Befehl	***Beschreibung***	***Anmerkung***
Tablecopy 50, 10	*10 Bytes von LUT ins RAM kopieren:* *LTU Startadresse: 50 bis 59* *RAM Zieldadresse: 50 bis 59*	*Ok*
Tablecopy 500, b1	*b1 Bytes von LUT ins RAM kopieren:* *LUT Startadresse: 50 bis 50 + b1* *RAM Zieldadresse: 50 bis 50 + b1*	*Ok*

2 Ausnahme sind die 08M2 Typen, die keine LUT beinhalten.

Befehl	*Beschreibung*	*Anmerkung*
Tablecopy w0, w1	*w1 Bytes von LUT ins RAM kopieren: LUT Startadresse: w0 bis w0 + w1 RAM Zieldadresse: w0 bis w0 + w1*	*Ok*
Tablecopy 1000, 10	*101 Bytes von LUT ins RAM kopieren: LUT Startadresse: 1000 bis 1009 RAM Zieldadresse: 1000 bis 1009*	*Befehl wird ignoriert, da sich Adresse 1000 außerhalb des verfügbaren LUT Speicher befindet*
Tablecopy 511, 10	*10 Bytes von LUT ins RAM kopieren: LUT Startadresse: 511 bis 520 RAM Zieldadresse: 511 bis 520*	*Nur das Byte an Adresse 511 wird kopiert. Die restlichen Bytes liegen außerhalb des verfügbaren LUT Speicher*

Tabelle 21

Readtable

Mittels Readtable können einzelne Bytes aus der LUT gelesen und in einer Variablen gespeichert werden. Der Syntax des Readtable-Befehls lautet:

Readtable *Adresse*, { *Byte-Variable* | *Word-Variable* } [, { … }]

Adresse bezeichnet eine allgemeine Variable, Systemvariable oder Konstante, welche die Speicheradresse in der LUT enthält, die das zu lesenden Byte beinhaltet.
Als Adresse kann Grundsätzlich jeder beliebige Wert angegeben werden. Liegt die Speicheradresse außerhalb des verfügbaren LUT Speichers, wird der Befehl ignoriert. Bei einem Programmtest innerhalb der Entwicklungsumgebung wird dabei keine Fehlermeldung ausgegeben.

Das zu lesende Byte kann in einer Byte-Variablen oder Word-Variablen gespeichert werden. Als Word-Variable kann entweder eine allgemeine Variable oder eine Systemvariable verwendet werden. Da mit dem Readtable Befehl nur einzelne Bytes gelesen werden können, enthält eine Word-Variable nach dem lesen nur das Byte an der entsprechenden LUT-Adresse.

Sollen mehrere Variablen mit dem Readtable Befehl gelesen werden, müssen diese durch ein Komma voneinander getrennt angegeben werden.

Im folgenden noch ein paar Beispiele:

Befehl	*Beschreibung*	*Anmerkung*
Readtable 20, b1	*Inhalt von Adresse 20 nach b1 kopieren*	*Ok*
Readtable 55, b1, b2, b3	*Inhalt von Adresse 55, 56 und 57 nach b1, b2 und b3 kopieren*	*Ok*
Readtable b0, b1	*Inhalt von Adresse die in b0 steht, nach b1 kopieren*	*Ok*

Befehl	***Beschreibung***	***Anmerkung***
Readtable 20, w0	*Inhalt von Adresse 20 nach w0 kopieren*	*Ok*
Readtable 100, s_w1	*Inhalt von Adresse 100 nach Systemvariable s_w1 kopieren*	*Ok*
Readtable w1, word w1	*Inhalt von Adresse die in w1 steht, nach w1 kopieren*	*Ok, die Anweisung **word** wird ignoriert*
Readtable base, Speed	*Inhalt von Adresse die in base steht, nach Variable Speed kopieren.*	*Ok, sofern Speed eine Variable*
Readtable 10000, b1	*Inhalt von Adresse 10000 nach b1 kopieren*	*Befehl wird ignoriert, da Adresse 10000 außerhalb des verfügbaren LUT Speicher liegt*